

Efficient Sensor/Model Based On-Line Collision Detection for Planetary Manipulators

Chris Leger
Jet Propulsion Laboratory
cleger@robotics.jpl.nasa.gov

Abstract

Manipulator safeguarding is a critical need for operations on planetary rovers. The computing environment on a Mars rover is extremely limited, which necessitates a highly efficient collision checking algorithm. We present such an algorithm that uses the Oriented Bounding Box (OBB) and a new primitive called the Oriented Bounding Prism (OBP) to detect potential self-collisions and collisions with terrain objects sensed with the rover's on-board stereo cameras; the algorithm thus has both model-based and sensor-based components. We have implemented the algorithm on JPL's FIDO rover and have tested it under realistic field conditions. Performance analysis indicate this method is significantly faster than previously reported results in the literature, in addition to incorporating sensed geometry. The method is also being used on board NASA's twin Mars Exploration Rovers, which are scheduled to launch in 2003.

Motivation

Ensuring the safety of rover-mounted manipulators in planetary exploration applications is of great importance. At best, a manipulator collision results in a lost day of operations due to latencies in communicating the failure back to Earth-based operators. At worst, the manipulator may become stuck in a deployed configuration or may damage the rover's stereo cameras, making further driving extremely difficult or impossible. Safeguarding for planetary rovers also presents a set of challenges not present in most other manipulator applications. Obviously, there can be no human observer or supervisor to trigger an emergency stop, and manipulator motions cannot be pre-programmed to guarantee safety since the manipulator target and surrounding obstacles are not known in advance. Finally, a highly time- and memory-efficient approach that does not impose significant operational delays is required due to the restricted computational resources of planetary rovers: for example, NASA's 2003 Mars Exploration Rovers (MER) mission will use 12MHz RAD6000 processors. While manipulator commands can will be checked for potential collisions before being sent to the rover, on-board safeguarding can catch errors introduced in the command sequencing or communication process, or operational errors not detected by pre-checking of individual trajectories (e.g. accidentally commanding the arm to move from a stowed to deployed position when it is already in a de-

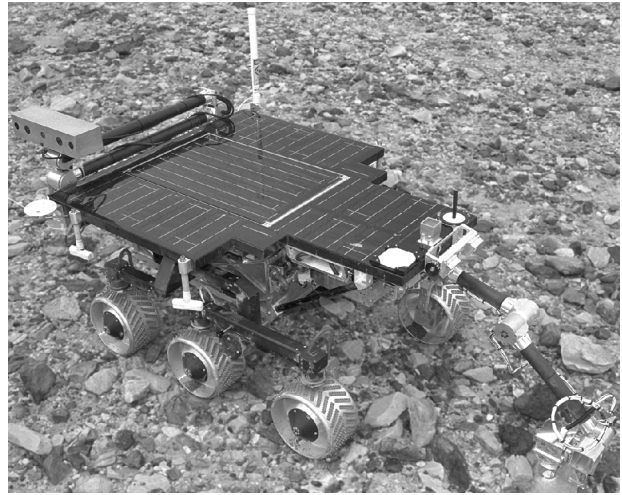


Figure 1: JPL FIDO rover during field testing.

ployed may cause a self-collision). On-board safeguarding is also essential for autonomous arm motions, which rule out the possibility of manual a priori safety checking of the arm trajectory.

This paper describes a model- and sensor-based method for preventing manipulator self-collisions and collisions with external objects. The method is efficient enough in terms of both time and memory to be used on-board robots with limited computational resources. We have tested the approach on the Jet Propulsion Lab's FIDO rover (Figure 1; see [Huntsberger99]) for both safeguarding of manually-commanded trajectories and fully autonomous arm deployments. We have also ported the algorithm to the MER flight software environment, and it will be used on-board the Mars Exploration Rovers during their surface operations in 2004.

Related Work

Historically, there has been significant interest in planning algorithms for generating collision-free trajectories given a model of the robot and obstacles (e.g. [Barraquand92]), including on-line but non-optimal approaches ([Khatib86]). The less complex problem of efficient on-line collision detection for manipulators has received less attention from the robotics community. Two broad categories of approaches are model-based and sensor-based, with sensor-based methods being further categorized as reactive or predictive. Both model-based and

predictive sensor-based methods explicitly model the robot and obstacles, with sensor-based approaches typically relying on LIDAR or stereo cameras to build the obstacle model. [Bon96] describes a model-based on-line approach for a telerobotic application: the manipulator is represented as a series of line segments with non-zero thickness, and a priori obstacles in the environment are represented as polyhedra. One reactive sensor-based approach, described in [Feddema94], uses capacitive proximity sensors to directly modify commanded manipulator motions to reduce or eliminate the motion of a link towards a sensed object. Greenspan and Burtnyk describe an on-line, constant-time approach suitable for sensor- or model-based use [Greenspan97], though this approach is inappropriate for our application due to difficulty in efficiently checking manipulator self-collisions and higher memory requirements (the 3D lookup table requires 1MB of storage and grows exponentially with desired resolution and linear dimension). Another voxel-based approach to collision detection (not specific to manipulators) is described in [Garcia94]. Previous work by the author of this paper was specialized to a robotic excavator with substantial dynamic effects, taking advantage of the workspace and manipulator geometry to simplify collision detection [Leger98]. This approach is not suitable for our application due to the limited manipulator representation and inability to check for self-collisions. More recently, [Hartman01] addressed interference prevention for two manipulators with significant dynamic effects, modeling the manipulator links with Oriented Bounding Boxes (OBBs; [Gottschalk96]), which we also use in the work described in this paper.

The primary differences between previous approaches and ours are as follows: both model-based (for self-collisions) and sensor-based (for external obstacles) collision tests must be included, and the method must be very fast and memory-efficient to meet the restrictions of the rover's on-board computer.

Oriented Bounding Boxes and Prisms

The heart of any manipulator collision detection algorithm is a means of checking two primitives (shapes) for collisions. While there exist a number of methods for detecting collisions between three-dimensional objects (e.g. [Canny86], [Gilbert88], [Green94]), the Oriented Bounding Box appears to offer the best trade-off of speed and accuracy for our application. A key advantage of OBBs is that no divisions, transcendental operations, or iterations are required to determine if two OBBs overlap; this leads to an extremely fast and robust implementation. In this work, we derive a modified OBB (called an Oriented Bounding Prism) and related intersection tests that efficiently model cylindrical geometry, since our rover's geometry is more accurately represented by a collection of cylinders and boxes than by boxes alone. In contrast to the

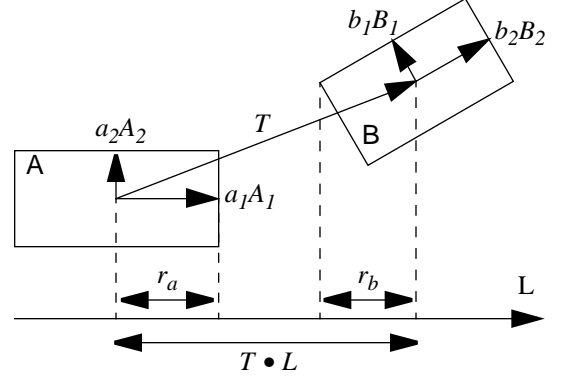


Figure 2: Separating line test for two OBBs

In this example, L is a separating line and is parallel to A_1 , the Y-axis of OBB A. B has been transformed to be relative to A's coordinate system. T is the location of B's center relative to A.

automatically-derived hierarchy of OBBs (called an OBB-Tree) terminating in triangular faces, we use a more restricted hierarchy of OBBs whose lowest level (highest detail) is a set of OBBs rather than triangles.

The standard OBB intersection tests (from [Gottschalk96]) are based on a theorem stating that for any two convex polyhedra, if the polyhedra do not overlap then there will be a plane of separation whose normal (called the *separating line*) is either parallel to at least one face of either polyhedra, or perpendicular to one edge of each polyhedra. Thus, to determine whether two OBBs overlap, one checks all possible normals of the separating plane (there are 15 cases) by projecting the extents of the OBBs onto each potential separating line (Figure 2). If the sum of the projected extents is greater than the sum of the projected radii of the OBBs plus a tolerance ϵ (the minimum "safe" separation), then the two OBBs are non-overlapping; otherwise, the remaining lines of separation must be tested. The general rule for determining if two OBBs A and B overlap is:

A and B are separated along the vector L iff

$$|T \cdot L| > r_a + r_b + \epsilon \quad (1)$$

where

$$r_a = \sum_{j=0}^2 |a_j A_j \cdot L| \quad (2)$$

$$r_b = \sum_{j=0}^2 |b_j B_j \cdot L| \quad (3)$$

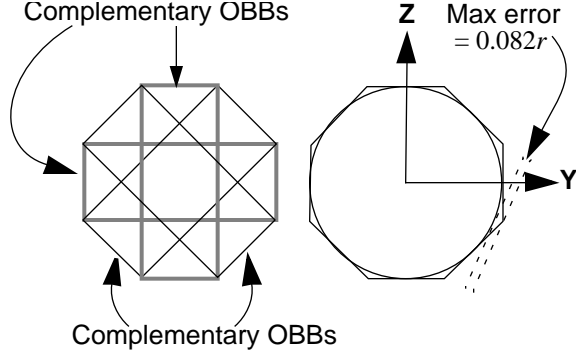


Figure 3: An OBP approximation ($n=8$) to a cylinder viewed end-on (i.e. along the X axis)

and ϵ is a safety tolerance. (See “Rover Model” below for a discussion on choosing an appropriate value for ϵ .) These tests can be made more computationally efficient if \mathbf{B} is transformed into \mathbf{A} ’s coordinate system: considering the restrictions on \mathbf{L} (it must be A_i , B_j , or $A_i \times B_j$ where $i, j \in \{0, 1, 2\}$), many of the vector terms reduce to scalars. For example, in the case where $\mathbf{L} = A_0 \times B_2$, the equations reduce to:

$$|\mathbf{T} \cdot \mathbf{L}| = T_2 B_{21} - T_1 B_{22} \quad (4)$$

$$r_a = a_1 |B_{22}| + a_2 |B_{21}| \quad (5)$$

$$r_b = b_0 |B_{10}| + b_1 |B_{00}| \quad (6)$$

where B_{ij} indicates the j^{th} element of \mathbf{B} ’s i^{th} axis. In the worst case (that is, when two OBBs are intersecting) the intersection tests require approximately 200 multiplications and additions, though the typical operation count is significantly lower [Gottschalk96].

Our modified OBBs for cylinders, which we call Oriented Bounding Prisms (OBPs), are based on the observation that an n -sided regular prism approximating a cylinder can be represented by $n/2$ overlapping, concentric OBBs, where n is an integer multiple of 4. We define two *complementary* OBBs \mathbf{A} and \mathbf{A}_c as being concentric and having the same dimensions but with one OBB offset by a 90 degree rotation about the X axis. Equivalently, \mathbf{A} and \mathbf{A}_c have the same center and orientation but have their Y and Z dimensions interchanged. There are $n/4$ pairs of complementary OBBs in an n -sided OBP (Figure 3). Each of the OBBs and its complement share many subexpressions in equations (4)-(6): the intersection tests for the a complement OBB are performed simply by swapping the Y and Z dimensions of one or the other OBBs. By interleaving the intersection tests for one OBB and its complement, the total number of calculations can be significantly reduced. The

calculation of $|\mathbf{T} \cdot \mathbf{L}|$ does not depend on any OBB dimensions, and is thus identical in the overlap tests for an OBB and its complement, and parts of r_a and r_b are shared as well. To check \mathbf{A} and its complement for collision with \mathbf{B} , again using $\mathbf{L} = A_0 \times B_2$, the additional equation for the complement of \mathbf{A} is

$$r_a' = a_2 |B_{22}| + a_1 |B_{21}| \quad (7)$$

and there is an additional comparison to determine if $|\mathbf{T} \cdot \mathbf{L}|$ is greater than $r_a' + r_b$. When \mathbf{A} is a single OBB and both \mathbf{B} and its complement are to be checked, there is a corresponding equation for r_b' in which b_1 and b_2 are exchanged. There are separate routines for checking two OBBs for collisions, for checking \mathbf{A} and its complement against \mathbf{B} , and for checking \mathbf{A} against \mathbf{B} and its complement. The complementary routines can be readily derived from the two-OBB routine by adding the complementary calculations (e.g. Equation (7)); the two-OBB routine can in turn be directly derived from Equations (1)-(3).

Our system uses octagonal prisms (i.e. $n=8$), yielding a maximum approximation error of 8.2% in the radial direction: that is, if an object is 0.082R away from actually contacting a cylinder of radius R, then a collision may be reported depending on the orientation of the approximated cylinder. The 8-sided approximation requires 4 OBBs at the lowest level, in addition to one higher-level OBB completely enclosing the cylinder. In contrast, using the OBB-Tree structure that ultimately represents each face would require at least 16 OBBs at the lowest level of the hierarchy (1 for each of 8 sides, and 4 for each end). Larger values of n can certainly be used, though one soon runs into diminishing returns: the maximum error for $n=4$ is 41.4%, for $n=8$ is 8.2%, for $n=12$ is 3.5%, and for $n=16$ is 2.0%. For our application, $n=4$ is too conservative, since there can sometimes be very little clearance during docking or final target approach. We will assume $n=8$ in the discussions in the remainder of the paper.

While checking two OBBs for intersections, or checking complementary OBBs against another OBB, can be performed using the equations described above, separate procedures are required to check OBP-OBB, OBP-OBP, and OBB-OBP pairs for collisions. (Note that the collision checks are order dependent, since \mathbf{B} is always transformed into \mathbf{A} ’s coordinate system.) The steps in testing OBP \mathbf{A} for collisions with OBB \mathbf{B} are:

Procedure 1: cylinderBoxCheck

- Compute the dimension $a_2' = a_1 \tan(\pi/8)$
- Check \mathbf{A}' (the OBB aligned with \mathbf{A} and having dimensions (a_0, a_1, a_2')) and its complement \mathbf{A}_c' against \mathbf{B} . Stop here if a collision is detected.
- Compute the OBB \mathbf{B}_{45} by rotating \mathbf{B} about \mathbf{A} ’s x-

- axis by 45°
- Check A' and A_c' , against B_{45}

The steps in testing OBB A for collisions with OBP B are similar:

Procedure 2: boxCylinderCheck

- Compute the dimension $b_2' = b_1 \tan(\pi/8)$
- Check A against B' (the OBB aligned with B and having dimensions (b_0, b_1, b_2')) and its complement. Stop here if a collision is detected.
- Compute the OBB B'_{45} by rotating B' about B's x-axis by 45°
- Check A against B'_{45} and its complement.

Finally, checking two OBPs for collisions uses both of the above procedures:

Procedure 3: cylinderCylinderCheck

- (Optional) Check the OBB enclosing A against the OBB enclosing B. Stop here if there are no collisions.
- Check A' and A_c' (computed as in *cylinderBox-Check*) against the OBB enclosing B.
- If A' had a collision, then check A' against the OBP for B using the procedure above. Stop here if collisions are detected.
- If A_c' had a collision, then check A_c' against the OBP for B using *boxCylinderCheck*. Stop here if collisions are detected. (At this point, we have checked A's axis-aligned sub-parts against all parts of B).
- Compute the OBB B'_{45} by rotating B' about A's x-axis by 45°
- Check A' and A_c' against the OBB enclosing B_{45} .
- If A' had a collision, then check A' against the OBP for B_{45} using the procedure above. Stop here if collisions are detected.
- If A_c' had a collision, then check A_c' against the OBP for B_{45} using *boxCylinderCheck*.

We use the same data structure for OBBs and OBPs, with a flag to indicate the object type. This facilitates the checking of an OBB as a faster first test for cylinder-box and cylinder-cylinder collisions, since the usual OBB-OBB test can be used directly. Only in cases of near-collision do the more involved OBP tests need to be performed. It is possible that further reduction in the average number of total operations can be reduced by eliminating some of the initial tests (e.g. the OBB approximation to cylinders that is checked before the OBP) based on runtime analyses of typical cases.

Note that the rotations and the use of the tangent

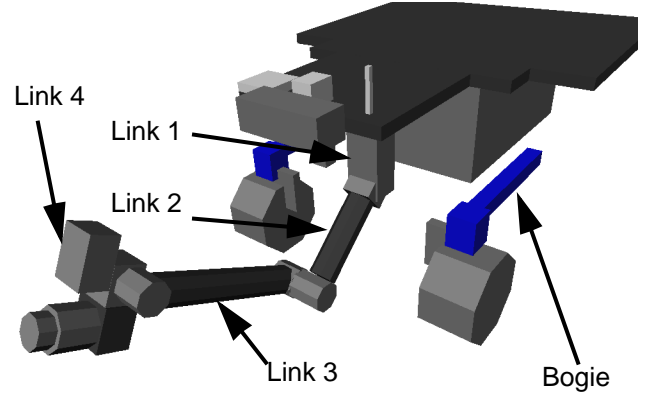


Figure 4: Models of the rover body, suspension, and manipulator. The rover suspension and steering joints are in their zero positions.

function above depend only on constants, and are hard-coded; no transcendental functions are used during runtime.

Rover Model

The rover's geometry is represented by a hand-built hierarchical model of OBBs and OBPs. Each of the 4-DOF manipulator's links has one high-level OBB encompassing all of the link's geometry, and the second through fourth links have lower-detail OBBs and OBPs representing more detailed geometry. The rover body is represented by several high-level OBBs and more detailed children. The front part of the rover's suspension is slightly more complex, since it is articulated. Each of the front wheel assemblies is represented by an OBP for the wheel and an OBB for the steering arm that can be moved to match the current steering angles. These parts, along with the bogie tube (the horizontal link leading towards the front wheel in Figure 4) and steering actuator housing are also affected by the rocker and bogie joint angles. All suspension parts for each side of the rover are enclosed in an OBB that moves with the rocker and bogie angles, but which is large enough to contain all parts for all steering angles. The rear portion of the suspension is not modeled since it is not in the manipulator's workspace.

Because of the kinematics of FIDO's manipulator, no self-collisions are possible within the links of the manipulator; however, self-collisions with other parts of the rover body or suspension are possible. Self-collisions are tested for between links 2, 3, 4 and the rover body and suspension parts; no self-collisions are possible for link 1. In some special cases, collisions between the robot's end effector are allowed: when docking, the end effector intentionally collides with a docking rig, and when placing instruments the end effector must contact the terrain (which will be described in the next section). For these cases, a flag

can be set indicating that the tip of the end effector should not be checked against the docking rig or terrain.

The safety tolerance ϵ in Equation 1 can be used to account for uncertainty in the rover model and motion control system. ϵ should be set to the sum of the uncertainties in the geometric knowledge for each OBB/OBP: for example, if a dimension for an OBB is only known to within 1mm, then ϵ for that part should be 1mm. This uncertainty could be due to limited knowledge of the model, error in joint angle measurements, or due to deflection of the links under load. ϵ can also be used to account for minor deviations between the planned and actual trajectory of the manipulator. The value of ϵ used in Equation 1 should be the sum of the ϵ values for each of the two OBBs/OBPs. For FIDO, we used a value of 2mm for all object pairs.

Multiresolution Terrain Model

The FIDO rover is equipped with several sets of stereo cameras, two of which (the “front HazCam” and “BellyCam”) image different parts of the manipulator workspace. For manipulator safeguarding, the raw stereo data must be transformed into a representation that allows efficient checking of collisions between manipulator and terrain objects. Given the typical operating environment of the robot—roughly horizontal terrain covered in rocks up to 30cm in size—an elevation map is an efficient and reasonably accurate representation for objects in the arm workspace. The height of each grid cell in the initial elevation map is the maximum height of all stereo data points (from both the BellyCam and HazCam) that lie within the cell’s bounds in the horizontal plane. A default elevation is used to fill in regions of the elevation map that are either occluded or are outside the sensors’ fields of view.

A multiresolution pyramid of elevation maps is then built from this first, highest-resolution elevation map. Each successively coarser map has half the linear resolution of the previous map, and the height of each grid cell is the height of the four higher-resolution grid cells encompassed by the lower-resolution grid cell.

As with rover objects, OBBs are used to represent terrain geometry. However, since the elevation maps are aligned with the rover frame and only one manipulator-terrain test is performed at a time, we do not need to explicitly create and store OBBs for each terrain grid cell. Instead, we use a single OBB and set the dimensions and center based on the grid cell being checked. The orientation of the OBB remains constant since the grid cells are always oriented parallel to the rover’s coordinate frame.

Figure 5 shows the OBBs involved in a collision between the terrain and part of the manipulator’s end effector. The terrain elevation map is shown as a grid of points. The highest-resolution terrain OBB that is colliding with the end effector is shown as a tall, thin, solid box, and the hierarchy of lower-level terrain OBBs that contain the col-

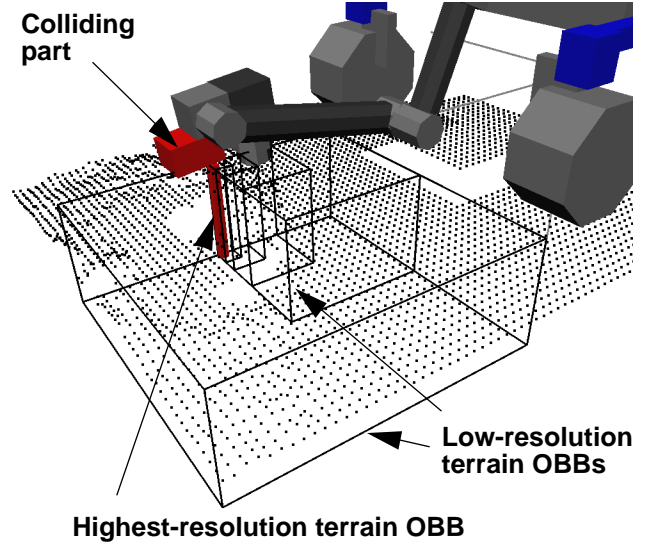


Figure 5: Example collision configuration. The lowest-resolution OBB enclosing the entire elevation map is omitted for clarity.

liding cell are shown as outlines.

System Integration

There were several candidate methods for integrating collision checking with the on-board rover software and the operations (off-board sequencing) software:

- Continuously check for arm collisions as the arm is moving.
- Check each commanded motion in the low-level rover software as the command is received
- Check each higher-level command (e.g. deploy arm, move to target) when the command is formulated (either on the ground by an operator, or on the rover for autonomous arm deployments)

We use a combination of the latter two approaches for several reasons. First, the rover controller checks that the arm accurately follows commanded trajectories, so checking trajectories a priori, rather than during execution, offers a high degree of safety. Second, it is preferable to know the rover is in a safe state if a collision is detected. Thus, preventing an arm deployment in the first place is preferred over halting the arm in a deployed state just short of a collision. Third, doing collision checking “on the ground” (i.e. as part of sequence planning by the human operators) leads to significant time savings by eliminating the entire abort-contingency-replan cycle involved in the execution of a failed command.

On-board safeguarding is implemented by checking each command for collisions in the routines for absolute and relative joint-space motion that are used by all higher-level arm control code. The current and goal joint angles are computed, and the trajectory is uniformly sam-

pled in joint-space such that samples are spaced less than one degree apart for the joint with the largest commanded motion. (Note that all DOFs are controlled to start and end their motions at the same time regardless of commanded joint motions.) Each pose along the trajectory is then checked for collisions; if any are detected, then the motion is not performed and the calling routine is informed of the failure. The routine for autonomous arm deployments also checks the entire sequence (deploy, move to standoff via point, place end effector, return to standoff, and stow) before commanding any single part of the trajectory.

Experiments and Performance

The initial testing phase of the collision avoidance software involved using it as a warning system for FIDO's safety observers: if the operators commanded a motion that would cause a collision, an audible alarm was sounded to alert the safety observer that a manual abort might be required. As errors in the rover model were fixed and the team gained confidence, the software was used to automatically halt dangerous motions, coming in particularly handy while debugging new, higher-level algorithms. We conducted field trials in May 2001 to train the 2003 MER project scientists in rover-based geology, and extensively used the collision checking software during rover operations planning (every commanded arm motion was checked and confirmed safe before being sent) and as an audible alert. At the time of the field trials, the MER rover was not slated to do on-board collision checking, so we only used the audible alert on FIDO; however, the software has since been incorporated into the MER on-board software. No false negatives (undetected collisions) were observed during our field testing, though there were some false positives during stowing and deployment of the arm, which involves small clearances between arm and rover objects. This is not a serious problem for FIDO or MER, since the stow and deploy sequences are hard-coded and have been manually verified to be free of collisions.

We have not noticed any significant operational delay while running the software on-board FIDO, and this observation is supported by performance measurements we have made on a desktop workstation (for the FIDO model), and on the MER flight software testbed, a 12 MHz RAD6000 (for the MER model). Repeated tests of a trajectory that has a collision between the end effector and terrain yield the following numbers, on an 800MHz Pentium III (roughly 4-6 times faster than FIDO's computer):

- Elevation map building (not counting stereo range map generation): 10ms
- Checking entire sequence (6 trajectories) for collisions: 2.4ms
- 34.7 arm poses checked per trajectory: 12 μ s per arm pose
- 18.2 primitive-primitive checks per arm pose:

0.64 μ s per primitive.

The key numbers are 12 μ s per arm pose and 2.4ms per sequence (in this case, deploying the arm from a stowed position to a target on the terrain). For the MER testbed computer and rover model, we measured the average time to check eight different arm poses for collisions (including terrain collisions). Several of the poses used had collisions in the end effector (the last link checked), while the others did not. The average time to test a pose for collisions was 5.4ms, and an average of 48 primitive-to-primitive collision tests were required for each pose. The MER model has 81 objects (about twice as many as the FIDO model), and unlike the FIDO model, the manipulator links must be checked for collisions with each other since the MER arm is capable of self-collisions.

In comparison, [Bon96] reports a time of 1.23ms for a model-based approach with 5 manipulator objects, running on a 100MHz R4600; [Greenspan96] reports a time of "less than 10ms" on a 66MHz 486, with 15s required for building the voxel map from a priori object models. No performance measurements were reported in [Hartman01], though their implementation is also likely to be fast due to the use of OBBs. While direct quantitative performance comparisons cannot be made without using the same hardware, compiler, and arm and obstacle objects, our performance measurements lead us to believe that our algorithm offers substantially decreased computational complexity.

Memory requirements for the algorithm are relatively small: the multiresolution map (64x64 at the highest resolution) requires 36K, and the FIDO arm and rover model requires an additional 4.2K. The executable size is 56K when compiled for a Pentium with gcc, with both debugging symbols (-g) and optimization (-O2) enabled. It is worth noting that the memory requirements are far less than for voxel-based approaches: [Greenberg96] lists a size of 1MB for the voxel map, and both memory and computation time increase with the cube of the desired robot or environment resolution. Our method is also directly applicable to multi-manipulator systems: the articulated suspension effectively acts as two additional rover-mounted manipulators. This also stands in contrast to voxel-based approaches that must precompute a voxel map of the environment, thus having difficulty in efficiently checking for collisions between moving objects.

Limitations

While efficient and robust, our method does have some limitations. One is that, since the terrain representation involves reducing a 3D surface to a 2.5D elevation map, concavities in the 3D surface that are not aligned with the axis of projection cannot be accurately represented. For example, imagine a small 'cave' in front of the rover. The elevation map created with a vertical axis of projection

cannot represent the cave, only the surface above the cave. The significance of this limitation depends on the expected frequency of such situations in the rover's environment, which has been fairly low in our experience. The effect can be mitigated to some degree by aligning the axis of projection used in the terrain map with the optical axis of the rover's stereo cameras, since the cameras can only create 2.5D representations of the environment. In this way, concavities that can be sensed by the rover will be more accurately represented in the terrain map. The accuracy is best near the center of the field of view, and degrades so that at the edge of the field of view, the terrain OBBs are misaligned with respect to the terrain-camera axis by half the field of view.

Another limitation is that significantly more OBBs and/or OBPs can be required to represent rover objects that are not accurately modeled by boxes or cylinders; this is part of the reason why the MER rover model is more complex than the FIDO model. Finally, the amount of storage space required for the elevation map grows with the number of cells in the elevation map, which itself grows with the square of the linear extent of the map divided by the cell size.

Conclusion

We have presented a highly efficient and robust collision detection method suitable for implementation on robots with limited computing resources. The method can detect both self-collisions and collisions with sensed terrain, is directly applicable to multi-manipulator systems, and preliminary performance measurements suggest a significant speedup over methods previously reported in the literature. We have implemented and tested the method on the FIDO rover, and the method is being used on the 2003 Mars Exploration Rovers.

Acknowledgments

This work was carried out at Jet Propulsion Laboratory, California Institute of Technology, under contract with National Aeronautics and Space Administration. The author would like to thank the members of the FIDO team for their significant efforts in developing, maintaining, and running the FIDO rover as an extremely robust platform for technology development: Paul Schenker, Hrand Aghazarian, Eric Baumgartner, Yang Cheng, Tony Ganino, Mike Garrett, Terry Huntsberger, Brett Kennedy, Lee Magnone, Jeff Norris, Ashitey Trebi-Ollennu, and Eddie Tunstel.

References

[Barraquand92] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-22(2):224-241, March/April 1992.

[Bon97] B. Bon and H. Seraji. Real-time model-based obstacle detection for the NASA Ranger Telerobot. In *Pro-*

ceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, April 1997.

[Canny86] J. F. Canny. Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:200-209, 1986.

[Feddema94] J. Feddema and J. Novak. Whole arm obstacle avoidance for teleoperated robots, in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 3303-3309, San Diego, 1994.

[Garcia-Alonso94] A. Garcia-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, 13(3):36-43, 1994.

[Gilbert88] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, vol. RA-4:193-203, 1988.

[Gottschalk96] S. Gottschalk, M. C. Lin and D. Manoch. *OBB-Tree: A hierarchical Structure for Rapid Interference Detection*. Technical Report TR96-013, Department of Computer Science, University of North Carolina, Chapel Hill, 1996.

[Green94] N. Greene. Detecting intersection of a rectangular solid and a convex polyhedron. In *Graphics Gems IV*, pp. 74-82, Academic Press, 1994.

[Greenspan96] M. Greenspan and N. Burtnyk. Obstacle Count Independent Real-Time Collision Avoidance. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 1996, pp. 1073-1080. 4.

[Hartman01] L. Hartman. A real-time approach to the coordination of multiple manipulators. In *Proceeding of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001*, St-Hubert, Quebec, Canada, June 18-22, 2001.

[Huntsberger99] T. L. Huntsberger, E. T. Baumgartner, H. Aghazarian, Y. Cheng, P. S. Schenker, P. C. Leger, K. D. Iagnemma, and S. Dubowsky, "Sensor fused autonomous guidance of a mobile robot and applications to Mars sample return operations," in *Proc. SPIE*, Vol. 3839, Boston, MA, Sept. 1999.

[Khatib86] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, *The International Journal of Robotics Research*, Spring 1986, Volume 5, Number 1, pp. 90-98.

[Leger98] C. Leger, P. Rowe, J. Bares, S. Boehmke, A. Stentz. Obstacle detection and safeguarding for a high-speed autonomous hydraulic excavator, In *Proceedings of SPIE Vol 3525*, Boston, MA 1998.